Operating Systems Notes

A computer system has many resources (hardware and software), which may be required to complete a task. The commonly required resources are input/output devices, memory, file storage space, CPU, etc. The operating system acts as a manager of the above resources and allocates them to specific programs and users, whenever necessary to perform a particular task. Therefore the operating system is the resource manager i.e. it can manage the resource of a computer system internally. The resources are processor, memory, files, and I/O devices.

In simple terms, an operating system is an interface between the computer user and the machine. It is very important for you that every computer must have an operating system in order to run other programs. The operating system mainly coordinates the use of the hardware among the various system programs and application programs for various users.

An operating system acts similarly like government means an operating system performs no useful function by itself; though it provides an environment within which other programs can do useful work. Below we have an abstract view of the components of the computer system:

- The Computer Hardware contains a central processing unit (CPU), the memory, and the input/output (I/O) devices and it provides the basic computing resources for the system.
- The Application programs like spreadsheets, Web browsers, word processors, etc. are used to define the ways in which these resources are used to solve the computing problems of the users. And the System program mainly consists of compilers, loaders, editors, OS, etc.
- The Operating System is mainly used to control the hardware and coordinate its use among the various application programs for the different users.
- Basically, Computer System mainly consists of hardware, software, and data.

OS is mainly designed in order to serve two basic purposes:

1. The operating system mainly controls the allocation and use of the computing System's resources among the various user and tasks.
2. It mainly provides an interface between the computer hardware and the programmer

that simplifies and makes feasible for coding, creation of application programs and debugging.

Two Views of Operating System

1. **User's View**: The user view of the computer refers to the interface being used. Such systems are designed for one user to monopolize its resources, to maximize the work that the user is performing. In these cases, the operating system is designed mostly for ease of use, with some attention paid to performance, and none paid to resource utilization.
2. **System View**: The operating system can be viewed as a resource allocator also. A computer system consists of many resources like - hardware and software - that must be managed efficiently. The operating system acts as the manager of the resources, decides between conflicting requests, controls the execution of programs, etc.

Types of Operating System

Given below are different types of Operating System:

1. Simple Batch System: In a Batch Operating System, the similar jobs are grouped together into batches with the help of some operator and these batches are executed one by one. For example, let us assume that we have 10 programs that need to be executed. Some programs are written in C++, some in C and rest in Java. Now, every time when we run these programmes individually then we will have to load the compiler of that particular language and then execute the code. But what if we make a batch of these 10 programmes. The benefit with this approach is that, for the C++ batch, you need to load the compiler only once. Similarly, for Java and C, the compiler needs to be loaded only once and the whole batch gets executed.

Advantages:

1. The overall time taken by the system to execute all the programmes will be reduced.
2. The Batch Operating System can be shared between multiple users.

Disadvantages:

1. Manual interventions are required between two batches.
2. The CPU utilization is low because the time taken in loading and unloading of batches is very high as compared to execution time.

2. Multiprogrammed: Sharing the processor, when two or more programs reside in memory at the same time, is referred as multiprogramming. Multiprogramming assumes

a single shared processor. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

An OS does the following activities related to multiprogramming:

- The operating system keeps several jobs in memory at a time.
- This set of jobs is a subset of the jobs kept in the job pool.
- The operating system picks and begins to execute one of the jobs in the memory.
- Multiprogramming operating systems monitor the state of all active programs and system resources using memory management programs to ensures that the CPU is never idle, unless there are no jobs to process.

Advantages:

- High and efficient CPU utilization.
- User feels that many programs are allotted CPU almost simultaneously.

Disadvantages:

- CPU scheduling is required.
- To accommodate many jobs in memory, memory management is required.

3. Time-Shared: In a Multi-tasking Operating System, more than one processes are being executed at a particular time with the help of the time-sharing concept. So, in the time-sharing environment, we decide a time that is called time quantum and when the process starts its execution then the execution continues for only that amount of time and after that, other processes will be given chance for that amount of time only. In the next cycle, the first process will again come for its execution and it will be executed for that time quantum only and again next process will come. This process will continue.

Advantages:

1. Since equal time quantum is given to each process, so each process gets equal opportunity to execute.
2. The CPU will be busy in most of the cases and this is good to have case.

Disadvantages:

1. Process having higher priority will not get the chance to be executed first because the equal opportunity is given to each process.

4. Personal Computer: Personal computer operating system provides a good interface to a single user. Personal computer operating systems are widely used for word processing, spreadsheets and Internet access. Personal computer operating system are made only for personal. You can say that your laptops, computer systems, tablets etc. are your personal computers and the operating system such as windows 7, windows 10, android,

etc. are your personal computer operating system. And you can use your personal computer operating system for your personal purposes, for example, to chatting with your friends using some social media sites, reading some articles from internet, making some projects through microsoft powerpoint or any other, designing your website, programming something, watching some videos and movies, listening to some songs and many more.

5. Parallel: Parallel processing requires multiple processors and all the processor works simultaneously in the system. Here, the task is divided into subparts and these subparts are then distributed among the available processors in the system. Parallel processing completes the job on the shortest possible time.

All the processors in the parallel processing environment should run on the same operating system. All processors here are tightly coupled and are packed in one casing. All the processors in the system share the common secondary storage like the hard disk. As this is the first place where the programs are to be placed.

There is one more thing that all the processors in the system share i.e. the user terminal (from where the user interact with the system). The user need not to be aware of the inner architecture of the machine. He should feel that he is dealing with the single processor only and his interaction with the system would be the same as in a single processor,

Advantages:

1. It saves time and money as many resources working together will reduce the time and cut potential costs.
2. It can be impractical to solve larger problems on Serial Computing.
3. It can take advantage of non-local resources when the local resources are finite.
4. Serial Computing 'wastes' the potential computing power, thus Parallel Computing makes better work of the hardware.

Disadvantages:

1. It addresses such as communication and synchronization between multiple sub-tasks and processes which is difficult to achieve.
2. The algorithms must be managed in such a way that they can be handled in a parallel mechanism.
3. The algorithms or programs must have low coupling and high cohesion. But it's difficult to create such programs.
4. More technically skilled and expert programmers can code a parallelism-based program well.

6. Distributed Systems: These types of the operating system is a recent advancement in the world of computer technology and are being widely accepted all over the world and, that too, with a great pace. Various autonomous interconnected computers communicate

with each other using a shared communication network. Independent systems possess their own memory unit and CPU. These are referred to as loosely coupled systems or distributed systems. These system's processors differ in size and function. The major benefit of working with these types of the operating system is that it is always possible that one user can access the files or software which are not actually present on his system but some other system connected within this network i.e., remote access is enabled within the devices connected in that network.

Advantages of Distributed Operating System:

- Failure of one will not affect the other network communication, as all systems are independent from each other
- Electronic mail increases the data exchange speed
- Since resources are being shared, computation is highly fast and durable
- Load on host computer reduces
- These systems are easily scalable as many systems can be easily added to the network
- Delay in data processing reduces

Disadvantages of Distributed Operating System:

- Failure of the main network will stop the entire communication
- To establish distributed systems the language which is used are not well defined yet
- These types of systems are not readily available as they are very expensive

7. Real Time Systems: It is developed for real-time applications where data should be processed in a fixed, small duration of time. It is used in an environment where multiple processes are supposed to be accepted and processed in a short time. RTOS requires quick input and immediate response, e.g., in a petroleum refinery, if the temperate gets too high and crosses the threshold value, there should be an immediate response to this situation to avoid the explosion. Similarly, this system is used to control scientific instruments, missile launch systems, traffic lights control systems, air traffic control systems, etc.

This system is further divided into two types based on the time constraints:

Hard Real-Time Systems: These are used for the applications where timing is critical or response time is a major factor; even a delay of a fraction of the second can result in a disaster. For example, airbags and automatic parachutes that open instantly in case of an accident. Besides this, these systems lack virtual memory.

Soft Real-Time Systems: These are used for application where timing or response time is less critical. Here, the failure to meet the deadline may result in a degraded performance instead of a disaster. For example, video surveillance (cctv), video player, virtual reality, etc. Here, the deadlines are not critical for every task every time.

Advantages of real-time operating system:

- The output is more and quick owing to the maximum utilization of devices and system
- Task shifting is very quick, e.g., 3 microseconds, due to which it seems that several tasks are executed simultaneously
- Gives more importance to the currently running applications than the queued application
- It can be used in embedded systems like in transport and others.
- It is free of errors.
- Memory is allocated appropriately.

Disadvantages of real-time operating system:

- A fewer number of tasks can run simultaneously to avoid errors.
- It is not easy for a designer to write complex and difficult algorithms or proficient programs required to get the desired output.
- Specific drivers and interrupt signals are required to respond to interrupts quickly.
- It may be very expensive due to the involvement of the resources required to work.

It provides an environment for the program to execute. It also provides users with the services of how to execute programs in a convenient manner. The operating system provides some services to program and also to the users of those programs. The specific services provided by the OS are of course different.

Following are the common services provided by an operating system:

1. **Program execution:** An operating system must be able to load many kinds of activities into the memory and to run it. The program must be able to end its execution, either normally or abnormally. A process includes the complete execution of the written program or code. There are some of the activities which are performed by the operating system:
   - The operating system Loads program into memory
   - It also Executes the program
   - It Handles the program's execution
   - It Provides a mechanism for process synchronization
   - It Provides a mechanism for process communication
2. **I/O operations:** The communication between the user and devices drivers are managed by the operating system.

- I/O devices are required for any running process. In I/O a file or an I/O devices can be involved.
- I/O operations are the read or write operations which are done with the help of input-output devices.
- Operating system give the access to the I/O devices when it required.

3. **File system manipulation:**
   - The collection of related information which represent some content is known as a file. The computer can store files on the secondary storage devices. For long-term storage purpose. examples of storage media include magnetic tape, magnetic disk and optical disk drives like CD, DVD.
   - A file system is a collection of directories for easy understand and usage. These directories contain some files. There are some major activities which are performed by an operating system with respect to file management.
   - The operating system gives an access to the program for performing an operation on the file.
   - Programs need to read and write a file.
   - The user can create/delete a file by using an interface provided by the operating system.
   - The operating system provides an interface to the user creates/ delete directories.
   - The backup of the file system can be created by using an interface provided by the operating system.
4. **Communication:** In the computer system, there is a collection of processors which do not share memory peripherals devices or a clock, the operating system manages communication between all the processes. Multiple processes can communicate with every process through communication lines in the network. There are some major activities that are carried by an operating system with respect to communication.
   - Two processes may require data to be transferred between the process.
   - Both the processes can be on one computer or a different computer, but are connected through a computer network.
5. **Error detection:** An error is one part of the system that may cause malfunctioning of the complete system. The operating system constantly monitors the system for detecting errors to avoid some situations. This give relives to the user of the worry of getting an error in the various parts of the system causing malfunctioning.

   The error can occur anytime and anywhere. The error may occur anywhere in the computer system like in CPU, in I/O devices or in the memory hardware. There are

some activities that are performed by an operating system:

- The OS continuously checks for the possible errors.
- The OS takes an appropriate action to correct errors and consistent computing.

6. **Resource allocation:** When there are multiple users or multiple jobs running at the same time resources must be allocated to each of them. There are some major activities that are performed by an operating system:
   - The OS manages all kinds of resources using schedulers.
   - CPU scheduling algorithm is used for better utilization of CPU.
7. **Protection:** The owners of information stored in a multi-user computer system want to control its use. When several disjoints processes execute concurrently it should not be possible for any process to interfere with another process. Every process in the computer system must be secured and controlled.

   Operating system can be implemented with the help of various structures. The structure of the OS depends mainly on how the various common components of the operating system are interconnected and melded into the kernel. Depending on this we have following structures of the operating system:

Simple structure: Such operating systems do not have well defined structure and are small, simple and limited systems. The interfaces and levels of functionality are not well separated. MS-DOS is an example of such operating system. In MS-DOS application programs are able to access the basic I/O routines. These types of operating system cause the entire system to crash if one of the user programs fails.

Advantages of Simple structure:

- It delivers better application performance because of the few interfaces between the application program and the hardware.
- Easy for kernel developers to develop such an operating system.

Disadvantages of Simple structure:

- The structure is very complicated as no clear boundaries exists between modules.
- It does not enforce data hiding in the operating system.

Layered structure: An OS can be broken into pieces and retain much more control on system. In this structure the OS is broken into number of layers (levels). The bottom layer (layer 0) is the hardware and the topmost layer (layer N) is the user interface. These layers are so designed that each layer uses the functions of the lower level layers only. This simplifies the debugging process as if lower level layers are debugged and an error occurs during debugging then the error must be on that layer only as the lower level layers have already been debugged.

The main disadvantage of this structure is that at each layer, the data needs to be modified and passed on which adds overhead to the system. Moreover careful planning of the layers is necessary as a layer can use only lower level layers. UNIX is an example of this structure.

Advantages of Layered structure:

- Layering makes it easier to enhance the operating system as implementation of a layer can be changed easily without affecting the other layers.
- It is very easy to perform debugging and system verification.

Disadvantages of Layered structure:

- In this structure the application performance is degraded as compared to simple structure.
- It requires careful planning for designing the layers as higher layers use the functionalities of only the lower layers.

Process

A process is a program at the time of execution.

Differences between Process and Program

- Process: Process is a dynamic object
- Program: Program is a static object
- Process is sequence of instruction execution
- Process loaded in to main memory
- Program is a sequence of instructions
- Time span of process is limited
- Process is a active entity
- Program loaded into secondary storage devices
- Time span of program is unlimited
- Program is a passive entity

Process States:

When a process executed, it changes the state, generally the state of process is determined by the current activity of the process. Each process may be in one of the following states:

1. New: The process is being created.
2. Running : The process is being executed.
3. Waiting : The process is waiting for some event to occur.
4. Ready: The process is waiting to be assigned to a processor.
5. Terminated : The Process has finished execution.

Only one process can be running in any processor at any time, But many process may be in ready and waiting states. The ready processes are loaded into a "ready queue".

New → Ready: OS creates process and prepares the process to be executed, then OS moved the process into ready queue.

Ready → Running: OS selects one of the Jobs from ready Queue and move them from ready to Running.

Running → Terminated : When the Execution of a process has Completed, OS terminates that process from running state. Sometimes OS terminates the process for some other reasons including Time exceeded, memory unavailable, access violation, protection Error, I/O failure and so on.

Running → Ready : When the time slot of the processor expired (or) If the processor received any interrupt signal, the OS shifted Running → Ready State.

Running → Waiting : A process is put into the waiting state, if the process need an event occur (or) an I/O Device require.

Waiting → Ready : A process in the waiting state is moved to ready state when the event for which it has been Completed.

Process Control Block

Each process is represented in the operating System by a Process Control Block. It is also called Task Control Block. It contains many pieces of information associated with a specific Process.

- Process State
- Program Counter
- CPU Registers
- CPU Scheduling Information
- Memory – Management Information
- Accounting Information
- I/O Status Information

Process Control Block

- Process State: The State may be new, ready, running, and waiting, Terminated...
- Program Counter: indicates the Address of the next Instruction to be executed.
- CPU registers: include accumulators, stack pointers,
- CPU-Scheduling Info: includes a process pointer, pointers to scheduling Queues, other scheduling parameters etc.
- Memory management Info: includes page tables, segmentation tables, value of base and limit registers.
- Accounting Information: includes amount of CPU used, time limits, Jobs(or)Process numbers.
- I/O Status Information: Includes the list of I/O Devices Allocated to the processes, list of open files.

Threads

A process is divided into number of light weight process, each light weight process is said to be a Thread. The Thread has a program counter (Keeps track of which instruction to execute next), registers (holds its current working variables), stack (execution History).

Thread States:

- born State: A thread is just created.
- ready state: The thread is waiting for CPU.
- running: System assigns the processor to the thread.
- sleep: A sleeping thread becomes ready after the designated sleep time expires.
- dead: The Execution of the thread finished.

Thread

- Thread takes less time to create.
- Less time to terminate.
- Execution is very fast.
- It takes less time to switch b/w two threads.
- Communication b/w two threads is easy.
- Threads can share same memory area.
- System calls are not required.
- Threads are tightly coupled.
- Requires few resources to execute.

Multithreading

A process is divided into number of smaller tasks each task is called a Thread. Number of Threads within a Process execute at a time is called Multithreading.

If a program, is multithreaded, even when some portion of it is blocked, the whole program is not blocked. The rest of the program continues working If multiple CPU's are available.

Multithreading gives best performance. If we have only a single thread, number of CPU's available, No performance benefits achieved.

- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

- CODE- Contains instruction
- DATA- holds global variable
- FILES-opening and closing files
- REGISTER- contain information about CPU state
- STACK-parameters, local variables, functions

PROCESS SCHEDULING

CPU is always busy in Multiprogramming. Because CPU switches from one job to another job. But in simple computers CPU sit idle until the I/O request granted.

Scheduling is an important OS function. All resources are scheduled before use (cpu, memory, devices. . .).

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Scheduling Objectives

- Maximize throughput.
- Maximize number of users receiving acceptable response times.
- Be predictable.
- Balance resource use.
- Avoid indefinite postponement.
- Enforce Priorities.
- Give preference to processes holding key resources.

Scheduling Queues

People live in rooms. Process are present in rooms known as queues. There are 3 types:

1. Job queue: when processes enter the system, they are put into a job queue, which consists all processes in the system. Processes in the job queue reside on mass storage and await the allocation of main memory.
2. Ready queue: if a process is present in main memory and is ready to be allocated to cpu for execution, is kept in ready queue.

3. Device queue: if a process is present in waiting state (or) waiting for an i/o event to complete is said to being device queue. The processes waiting for a particular I/O device is called device queue.

Schedulers

There are 3 schedulers:

1. Long term scheduler.
2. Medium term scheduler.
3. Short term scheduler.

Scheduler duties:

- Maintains the queue.
- Select the process from queues assign to CPU.

Types of schedulers

1. **Long term scheduler:** select the jobs from the job pool and loaded these jobs into main memory (ready queue). Long term scheduler is also called job scheduler.
2. **Short term scheduler:** select the process from ready queue, and allocates it to the cpu. If a process requires an I/O device, which is not present available then process enters device queue. short term scheduler maintains ready queue, device queue. Also called as cpu scheduler.
3. **Medium term scheduler:** if process request an I/O device in the middle of the execution, then the process removed from the main memory and loaded into the waiting queue. When the I/O operation completed, then the job moved from waiting queue to ready queue. These two operations performed by medium term scheduler.

Context Switch

Assume, main memory contains more than one process. If cpu is executing a process, if time expires or if a high priority process enters into main memory, then the scheduler saves information about current process in the PCB and switches to execute the another process. The concept of moving CPU by scheduler from one process to other process is known as context switch.

Types of Scheduling

- Non-Preemptive Scheduling: CPU is assigned to one process, CPU do not release until the competition of that process. The CPU will assigned to some other process only

after the previous process has finished.

- Preemptive scheduling: here CPU can release the processes even in the middle of the execution. CPU received a signal from process p2. OS compares the priorities of p1 ,p2. If p1 > p2, CPU continues the execution of p1. If p1 < p2 CPU preempt p1 and assigned to p2.

Dispatcher

The main job of dispatcher is switching the cpu from one process to another process. Dispatcher connects the cpu to the process selected by the short term scheduler.

Dispatcher latency: The time it takes by the dispatcher to stop one process and start another process is known as dispatcher latency. If the dispatcher latency is increasing, then the degree of multiprogramming decreases.

Scheduling Criteria

- Throughput: how many jobs are completed by the cpu with in a time period.
- Turnaround time: The time interval between the submission of the process and time of the completion is turnaround time.
- Waiting time: The time spent by the process to wait for cpu to be allocated.
- Response time: Time duration between the submission and first response.
- CPU Utilization: CPU is costly device, it must be kept as busy as possible. Eg: CPU efficiency is 90% means it is busy for 90 units, 10 units idle.

CPU Scheduling Algorithms

1. **First come First served scheduling (FCFS):** The process that request the CPU first is holds the cpu first. If a process request the cpu then it is loaded into the ready queue, connect CPU to that process. It is Non Preemptive Scheduling Algorithm.

   Disadvantage: Average waiting time is very high.

2. **Shortest Job First Scheduling (SJF):** Which process having the smallest CPU burst time, CPU is assigned to that process. If two process having the same CPU burst time, FCFS is used.

   SJF is Non preemptive scheduling algorithm.

   Advantages:

   - Least average waiting time

- Least average turnaround time
- Least average response time
- Average waiting time (FCFS) = 25 ms
- Average waiting time (SJF) = 12.6 ms (50% time saved in SJF)

Disadvantages:

- Knowing the length of the next CPU burst time is difficult.
- Aging (Big Jobs are waiting for long time for CPU)

3. **Shortest Remaining Time First (SRTF):** This is preemptive scheduling algorithm. Short term scheduler always chooses the process that has term shortest remaining time. When a new process joins the ready queue, short term scheduler compares the remaining time of executing process and new process. If the new process has the least CPU burst time, The scheduler selects that job and connect to CPU. Otherwise continue the old process.
4. **Round Robin Scheduling Algorithm:** It is designed especially for time sharing systems. Here CPU switches between the processes. When the time quantum expired, the CPU switched to another job. A small unit of time, called a time quantum or time slice. A time quantum is generally from 10 to 100 ms. The time quantum is generally depending on OS. Here ready queue is a circular queue. CPU scheduler picks the first process from ready queue, sets timer to interrupt after one time quantum and dispatches the process.

   Disadvantage: Starvation.

   Starvation means only high priority process are executing, but low priority process are waiting for the CPU for the longest period of the time.

Multiple-Processor Scheduling

When multiple processes are available, then the scheduling gets more complicated, because there is more than one CPU which must be kept busy and in effective use at all times.

Load sharing resolves around balancing the load between multiple processors. Multi processor systems may be heterogeneous (It contains different kinds of CPU's) or Homogeneous (all the same kind of CPU).

1. **Approaches to multiple-processor scheduling**

   (a) Asymmetric multiprocessing: One processor is the master, controlling all activities and running all kernel code, while the other runs only user code.

(b) Symmetric multiprocessing: Each processor schedules its own job. Each processor may have its own private queue of ready processes.

2. **Processor Affinity**: Successive memory accesses by the process are often satisfied in cache memory. What happens if the process migrates to another processor? The contents of cache memory must be invalidated for the first processor; cache for the second processor must be repopulated. Most Symmetric multi processor systems try to avoid migration of processes from one processor to another processor, keep a process running on the same processor. This is called processor affinity.
   - Soft affinity: Soft affinity occurs when the system attempts to keep processes on the same processor but makes no guarantees.
   - Hard affinity: Process specifies that it is not to be moved between processors.
3. **Load balancing:** One processor won't be sitting idle while another is overloaded. Balancing can be achieved through push migration or pull migration.
   - Push migration: Push migration involves a separate process that runs periodically (e.g every 200 ms) and moves processes from heavily loaded processors onto less loaded processors.
   - Pull migration: Pull migration involves idle processors taking processes from the ready queues of the other processors.

DEADLOCKS

System model:

A system consists of a finite number of resources to be distributed among a number of competing processes. The resources are partitioned into several types, each consisting of some number of identical instances. Memory space, CPU cycles, files, I/O devices are examples of resource types. If a system has 2 CPUs, then the resource type CPU has 2 instances.

A process must request a resource before using it and must release the resource after using it. A process may request as many resources as it requires to carry out its task. The number of resources requested may not exceed the total number of resources available in the system. A process cannot request 3 printers if the system has only two.

A process may utilize a resource in the following sequence:

1. **REQUEST:** The process requests the resource. If the request cannot be granted immediately (if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
2. **USE:** The process can operate on the resource. If the resource is a printer, the process can print on the printer.

3. **RELEASE:** The process releases the resource.

For each use of a kernel managed by a process the operating system checks that the process has requested and has been allocated the resource. A system table records whether each resource is free (or) allocated. For each resource that is allocated, the table also records the process to which it is allocated. If a process requests a resource that is currently allocated to another process, it can be added to a queue of processes waiting for this resource.

To illustrate a deadlocked state, consider a system with 3 CDRW drives. Each of 3 processes holds one of these CDRW drives. If each process now requests another drive, the 3 processes will be in a deadlocked state. Each is waiting for the event "CDRW is released" which can be caused only by one of the other waiting processes. This example illustrates a deadlock involving the same resource type.

Deadlocks may also involve different resource types. Consider a system with one printer and one DVD drive. The process P_i is holding the DVD and process P_j is holding the printer. If P_i requests the printer and P_j requests the DVD drive, a deadlock occurs.

Deadlock Characterization

In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.

Necessary Conditions

A deadlock situation can arise if the following 4 conditions hold simultaneously in a system:

1. **Mutual Exclusion:** Only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
2. **Hold and Wait:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
3. **No Preemption:** Resources cannot be preempted. A resource can be released only voluntarily by the process holding it, after that process has completed its task.
4. **Circular Wait:** A set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes must exist such that P_0 is waiting for resource held by P_1, P_1 is waiting for resource held by P_2, $\ldots$, P_n is waiting for resource held by P_0.

Resource Allocation Graph

Deadlocks can be described more precisely in terms of a directed graph called a system resource allocation graph. This graph consists of a set of vertices V and a set of edges E. The set of vertices V is partitioned into 2 different types of nodes:

$P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the active processes in the system.

$R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.

A directed edge from process P_i to resource type R_j is denoted by $P_i \rightarrow R_j$. It signifies that process P_i has requested an instance of resource type R_j and is currently waiting for that resource.

A directed edge from resource type R_j to process P_i is denoted by $R_j \rightarrow P_i$, it signifies that an instance of resource type R_j has been allocated to process P_i.

A directed edge $P_i \rightarrow R_j$ is called a requested edge. A directed edge $R_j \rightarrow P_i$ is called an assignment edge.

We represent each process P_i as a circle, each resource type R_j as a rectangle. Since resource type R_j may have more than one instance, we represent each such instance as a dot within the rectangle. A request edge points to only the rectangle R_j. An assignment edge must also designate one of the dots in the rectangle.

When process P_i requests an instance of resource type R_j, a request edge is inserted in the resource allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource, it releases the resource, as a result, the assignment edge is deleted.

Sets:

- $P = \{P_1, P_2, P_3\}$
- $R = \{R_1, R_2, R_3, R_4\}$
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

One instance of resource type R_1, two instances of resource type R_2, one instance of resource type R_3, three instances of resource type R_4.

Process states:

Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of resource type R_1.

Process P_2 is holding an instance of R_1 and an instance of R_2 and is waiting for instance of R_3.

Process P_3 is holding an instance of R_3.

If the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist.

Suppose that process P_3 requests an instance of resource type R_2. Since no resource instance is currently available, a request edge $P_3 \rightarrow R_2$ is added to the graph.

Cycles:

$$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1, \quad P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$$

Processes P_1, P_2, P_3 are deadlocked. Process P_2 is waiting for the resource R_3, which is held by process P_3. Process P_3 is waiting for either process P_1 or P_2 to release resource R_2. In addition, process P_1 is waiting for process P_2 to release resource R_1.

There is also a cycle: $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$. However, there is no deadlock. Process P_4 may release its instance of resource type R_2. That resource can then be allocated to P_3, breaking the cycle.

Deadlock Prevention

For a deadlock to occur, each of the 4 necessary conditions must hold. By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

- Mutual Exclusion – not required for sharable resources; must hold for non sharable resources
- Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources. Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none. Low resource utilization; starvation possible
- No Preemption – If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released. Preempted resources are added to the list of resources for which the process is waiting. Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
- Circular Wait – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

Deadlock Avoidance

Requires that the system has some additional a priori information available. Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.

The deadlock-avoidance algorithm dynamically examines the resource allocation state to ensure that there can never be a circular-wait condition.

Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

Safe State

When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.

System is in safe state if there exists a sequence $\langle P_1, P_2, \ldots, P_n \rangle$ of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < i$.

That is:

- If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
- When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
- When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

If a system is in safe state no deadlocks.

If a system is in unsafe state possibility of deadlock.

Avoidance algorithms

Single instance of a resource type: Use a resource-allocation graph.

Multiple instances of a resource type: Use the banker's algorithm.

Resource Allocation Graph Scheme

Claim edge $P_i \rightarrow R_j$ indicated that process P_j may request resource R_j; represented by a dashed line.

Claim edge converts to request edge when a process requests a resource. Request edge converted to an assignment edge when the resource is allocated to the process. When a resource is released by a process, assignment edge reconverts to a claim edge. Resources must be claimed a priori in the system.

Unsafe State In Resource Allocation Graph

Banker's Algorithm

Multiple instances: Each process must a priori claim maximum use.

When a process requests a resource it may have to wait.

When a process gets all its resources it must return them in a finite amount of time.

Let n = number of processes, and m = number of resources types.

- Available: Vector of length m. If available $[j] = k$, there are k instances of resource type R_j available.
- Max: $n \times m$ matrix. If Max $[i, j] = k$, then process P_i may request at most k instances of resource type R_j.
- Allocation: $n \times m$ matrix. If Allocation$[i, j] = k$ then P_i is currently allocated k instances of R_j.
- Need: $n \times m$ matrix. If Need$[i, j] = k$, then P_i may need k more instances of R_j to complete its task.
- Need $[i, j]$ = Max $[i, j]-$ Allocation $[i, j]$

Safety Algorithm

1. Let Work and Finish be vectors of length m and n, respectively.
2. Initialize: Work = Available, Finish $[i]$ =false for $i = 0, 1, \ldots, n - 1$
3. Find an i such that both:
   (a) Finish $[i]$ = false
   (b) Need $i \leq$ Work

   If no such i exists, go to step 4
4. Work = Work + Allocation$_i$, Finish$[i]$ = true
5. Go to step 2
6. If Finish $[i]$ =true for all i, then the system is in a safe state

Resource-Request Algorithm for Process P_i

Request = request vector for process P_i. If Request$_i[j] = k$ then process P_i wants k instances of resource type R_j.

1. If $\text{Request}_i \leq \text{Need}_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $\text{Request}_i \leq$ Available, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:
   Available = Available - Request;
   $\text{Allocation}_i = \text{Allocation}_i + \text{Request}_i$;
   $\text{Need}_i = \text{Need}_i - \text{Request}_i$;
   If safe the resources are allocated to P_i; if unsafe, P_i must wait, and the old resource-allocation state is restored.

Deadlock Detection

Allow system to enter deadlock state. Detection algorithm.

Recovery scheme

- Process Termination: Abort all deadlocked processes. Abort one process at a time until the deadlock cycle is eliminated. In which order should we choose to abort?
  - Priority of the process
  - How long process has computed, and how much longer to completion
  - Resources the process has used
  - Resources process needs to complete
  - How many processes will need to be terminated
  - Is process interactive or batch?
- Resource Preemption: Selecting a victim – minimize cost
- Rollback – return to some safe state, restart process for that state
- Starvation – same process may always be picked as victim, include number of rollback in cost factor

Process Management And Synchronization

In a single processor multiprogramming system the processor switches between the various jobs until to finish the execution of all jobs. These jobs will share the processor time to get the simultaneous execution. Here the overhead is involved in switching back and forth between processes.

Critical Section Problem

Consider a system consisting of n processes $\{P_0, P_1, \ldots, P_{n-1}\}$. Each process has a segment of code called critical section in which the process may be changing common variables, updating a table, writing a file, etc. When one process is executing in its critical section, no other process is allowed into its critical section.

Design a protocol in such a way that the processes can cooperate each other.

Requirements:

- Mutual exclusion - Only one process can execute their critical sections at any time.
- Progress - If no process is executing in its critical section, any other process can enter based on the selection.
- Bounded waiting - Bounded waiting bounds on the number of times that the other processes are allowed to enter their critical sections after a process has made a request to enter into its critical section and before that the request is granted.

Synchronization With Hardware

Certain features of the hardware can make programming task easier and it will also improve system efficiency in solving critical-section problem.

For achieving this in uniprocessor environment we need to disable the interrupts when a shared variable is being modified. Whereas in multiprocessor environment disabling the interrupts is a time consuming process and system efficiency also decreases.

Syntax for interrupt disabling process:

```
repeat
disable interrupts;
critical section;
enable interrupts;
remainder section
forever
```

For eliminating the problems in interrupt disabling techniques particularly in multiprocessor environment, we need to use special hardware instructions.

In a multiprocessor environment, several processors share access to a common main memory. Processors behave independently. There is no interrupt mechanism between processors.

At a hardware level access to a memory location excludes any other access to that same location. With this foundation designers have proposed several machine instructions that

carry out two actions automatically such as reading and writing or reading and testing of a single memory location.

Most widely implemented instructions are:

Test-and-set instruction

Test-and-set instruction executes automatically. If two test-and-set instructions are executed simultaneously, each on a different CPU, then they will execute sequentially. That is access to a memory location excludes any other access to that same location which is shared one.

Implementation Code:

```
function Test-and-set (var i : integer) : boolean;
begin
if i = 0 then
begin
i := 1;
Test-and-set := true
end
else
Test-and-set := false
end.
```